

Real-Time Systems
Prof. Dr. Rajib Mall
Department of Computer Science and Engineering
Indian Institute of Technology, Kharagpur

Module No. # 01

Lecture No. # 23


A Few Basic Issues in Real-Time Operating Systems (Contd.)

So, let us get started with what we are doing last time. So, we are discussing the basic issues, few basic issues in real time operating systems.

(Refer Slide Time: 00:35)

Clock Interrupt Processing

- At each clock interrupt, the kernel carries out the following:
 - Process timer events
 - Update execution budget
 - Update the ready queue



03/08/10 504


We had halfway discussed about the clock interrupt processing. Let said that, whenever a clock interrupt occurs and these occurs very frequently, the kernel carries out the three main tasks, it checks whether any timer has expired and then queues the corresponding action as a task.

So, these are the three things, it processes the timer events for each timer. It checks whether it has expired and then queue is the corresponding action. It updates the executing execution budget of the currently executing task and then updates the ready queue.

(Refer Slide Time: 01:31)

Process Timer Events

- The timer queue:
 - Contains all timers arranged in order of their expiration times.
- At each clock interrupt:
 - The kernel checks if any timer event has occurred.
 - Kernel processes all timer events:
 - Queues the specified actions.



The diagram shows a horizontal bar representing a queue, divided into segments of red and green. Below the bar, there are three dots and two teardrop-shaped icons, one green and one red, hanging from the end of the queue.

03/08/10 506

In the processing timer event, we have to set that the timers are arranged in a queue, and each timer will be associated with a handler routine, and after the timer expires the handler routine is queued.

(Refer Slide Time: 01:53)

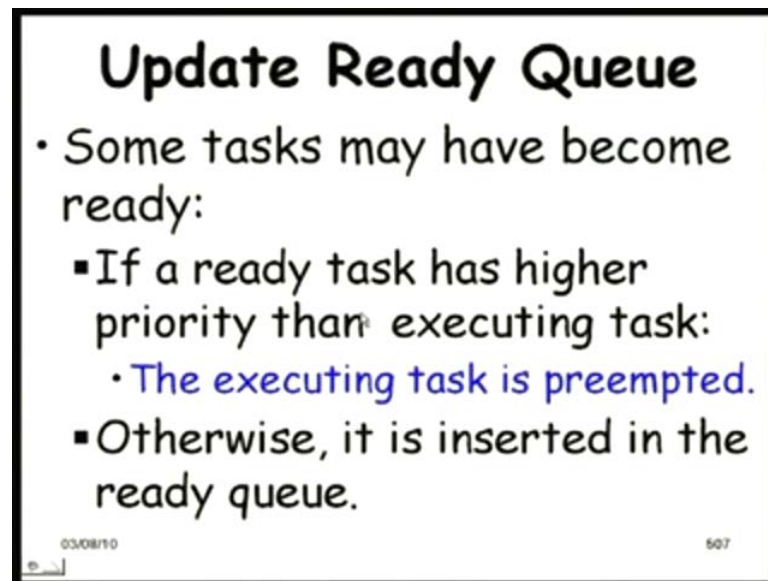
Update Execution Budget

- After each clock interrupt:
 - The scheduler decrements the remaining time slice of the executing task.
- If the task is not complete and the slice (budget) becomes 0:
 - The task is preempted.

03/08/10 506

And then we had said that after each clock interrupt the scheduler decrements the remaining time slice for the task, and then once the time slice become 0 the task is preempted.

(Refer Slide Time: 02:09)



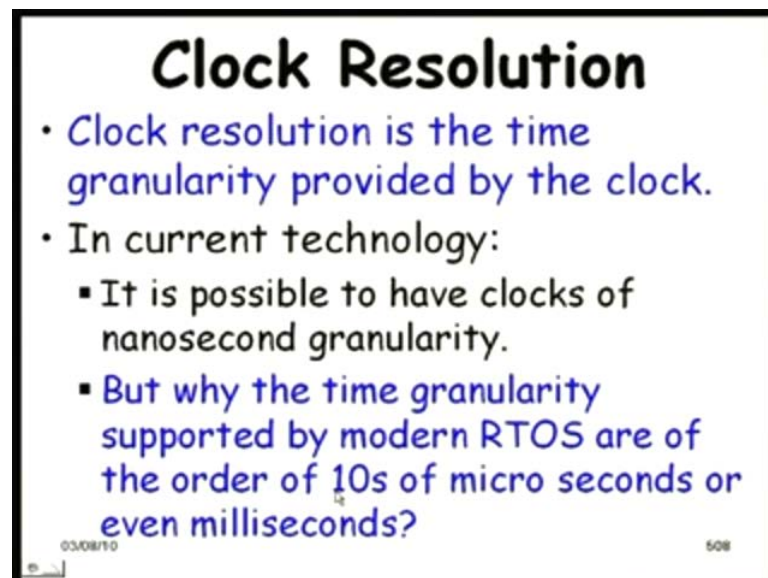
Update Ready Queue

- Some tasks may have become ready:
 - If a ready task has higher priority than executing task:
 - The executing task is preempted.
 - Otherwise, it is inserted in the ready queue.

03/08/10 607

Some events might have occurred since the clock last clock interrupt, and based on that actions are taken may be the currently executing task might have to be preempted if a higher priority task becomes ready.

(Refer Slide Time: 02:30)



Clock Resolution

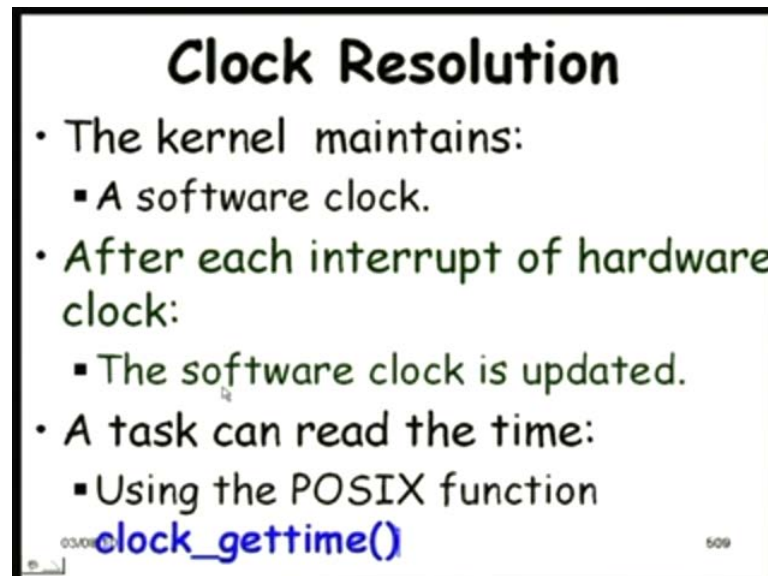
- Clock resolution is the time granularity provided by the clock.
- In current technology:
 - It is possible to have clocks of nanosecond granularity.
 - But why the time granularity supported by modern RTOS are of the order of 10s of micro seconds or even milliseconds?

03/08/10 608

Now, we are discussing about the clock resolution being supported by the operating system. We had said that the current hardware clock that is available is the gigahertz clock should be possible to have nanosecond clock, but the granularity that we will

discuss, now the subsequent lectures is microseconds several microseconds or even milliseconds and we are trying to examine the reason behind this.

(Refer Slide Time: 03:06)



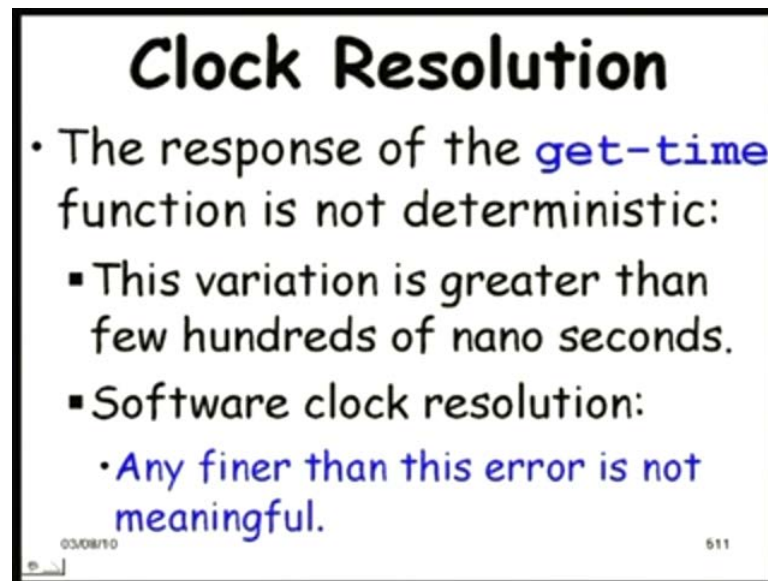
Clock Resolution

- The kernel maintains:
 - A software clock.
- After each interrupt of hardware clock:
 - The software clock is updated.
- A task can read the time:
 - Using the POSIX function `clock_gettime()`

OS/01/01 509

The reason is that the clock that we are referring to is a software clock, which is maintain based on the hardware interrupts, and then we are trying to answer the question that even the software clock, why cannot it be a nanosecond software clock. The answer is that, whenever we try to get the clock time we need to invoke a function don't directly read the clock time, because it is an operating system function, for example clock get time.

(Refer Slide Time: 03:54)



Clock Resolution

- The response of the `get-time` function is not deterministic:
 - This variation is greater than few hundreds of nano seconds.
 - Software clock resolution:
 - Any finer than this error is not meaningful.

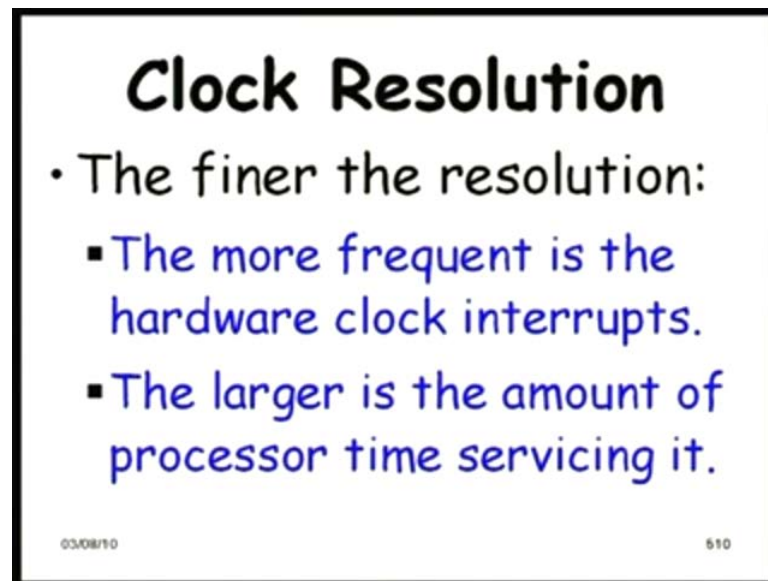
03/08/10 611

So, let me just discuss about that. So, the response to a function invocation is not deterministic, it is not that every time it will complete if we takes few clocks cycle to execute it, it is not that it will always execute in that much time, because it depends on what task currently running whether there are other interrupts and so on.

So, this variation in the completion of the get time function is much larger than several hundreds of nanoseconds. A software clock of any final resolution would not be meaningful, because the time that you read there is a variation in the time, if you are read it as some hundred nanosecond, it may be hundred ten, it may be hundred five, it may be hundred; so, there is a jitter there.

So, if you make the clock any finer than the jitter that is there in the get-time function, the clock is meaningless, you are maintaining a clock, but the time that you read is not exactly that time. So, that is one reason the other use reason is of course, as some of you are saying the overhead involved in maintaining the software clock.

(Refer Slide Time: 05:22)



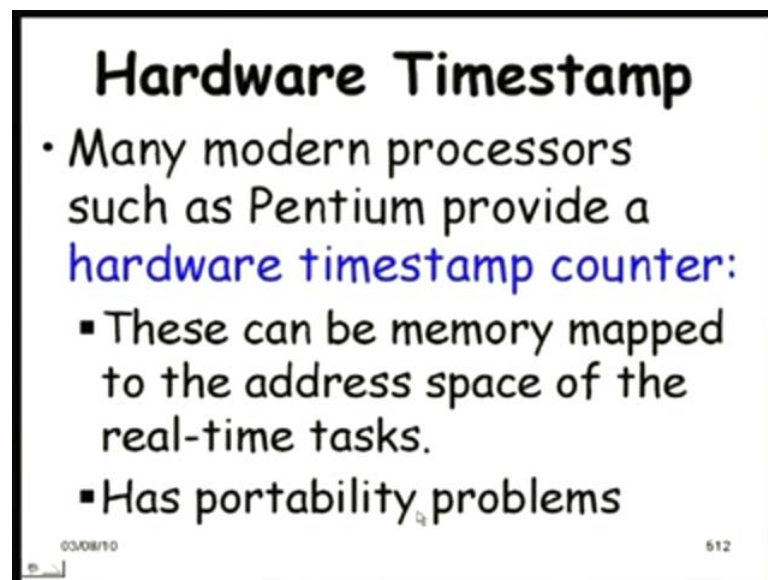
Clock Resolution

- The finer the resolution:
 - The more frequent is the hardware clock interrupts.
 - The larger is the amount of processor time servicing it.

03/08/10 610

The more frequent need to increment the software clock, the greater is the amount processor time servicing it.

(Refer Slide Time: 05:38)



Hardware Timestamp

- Many modern processors such as Pentium provide a hardware timestamp counter:
 - These can be memory mapped to the address space of the real-time tasks.
 - Has portability problems

03/08/10 612

But, some time you need to read the nanosecond clocks, there are some events which would require you to compare with the time that as elapsed up to a nanosecond accuracy, and we will see that many of these operating system, they do not really use the software clock to provide a nanosecond clock that can be read by the task.

But, what they use is a hardware timestamp counter while discussing the operating system we will just point that out, even while discussing the project standard we will discuss about this. The nanosecond clock time can be read through a memory mapped address space of the real-time tasks. Even for the other events when we use separate software and they cannot be rightly need to there must be hardware clock why separate something it should be (()) hardware time and hardware concept i mean.

So, let us look at the answer to that question that, why not always use this hardware time stamp. See, the answer is that we do not really, see that made give you a value of a 32 bit something, do not need that, you need the current time the time it has or you want to set a timer, we can directly use the hardware time for most of the applications, and the other problem with the hardware time stamp is that, it has portability problem.

Because, you directly using the hardware it is a memory mapped, if there is any change in the configuration use a different processor, all your applications need to change. So, this has a problem. So, unless the application requires it, do not use the hardware time stamp and this is also not supported by many of the operating system. Then how are the issues handled in which is some means hardware is. See, if you need a nanosecond granularity for some time application, the application will have portability problem, it will use the hardware clock and it is a difficult application.

(Refer Slide Time: 08:01)

Timer Services

- Timer service:
 - A vital service provided to applications by real-time operating systems.
- There are 2 types of timers:
 - Periodic
 - One shot (or aperiodic)

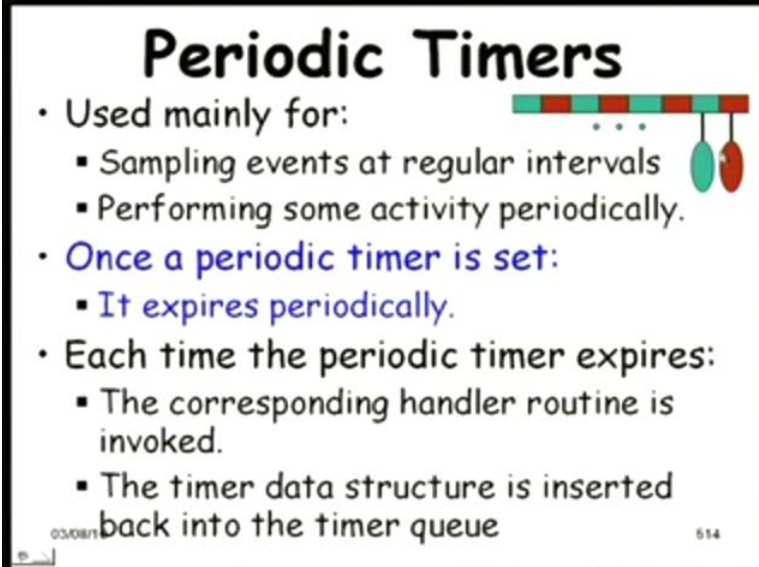
03/08/10

613

Now, let us look at the kind of timer services that will be provided by these operating systems we are going to discuss. The timer is a vital service for any real time operating system; actually we had said that this is one of the characteristic that differentiates the traditional operating systems with the real time operating systems.

Actually, all these real time operating systems that we will discuss, they will have two types of timers all these real time operating systems, each task can set several periodic timers and also several one shot or a periodic timer. So, if you look across the tasks then there will be many periodic timers and one shot timers that can be set.

(Refer Slide Time: 08:56)



Periodic Timers

- Used mainly for:
 - Sampling events at regular intervals
 - Performing some activity periodically.
- Once a periodic timer is set:
 - It expires periodically.
- Each time the periodic timer expires:
 - The corresponding handler routine is invoked.
 - The timer data structure is inserted back into the timer queue

03/08/17 514

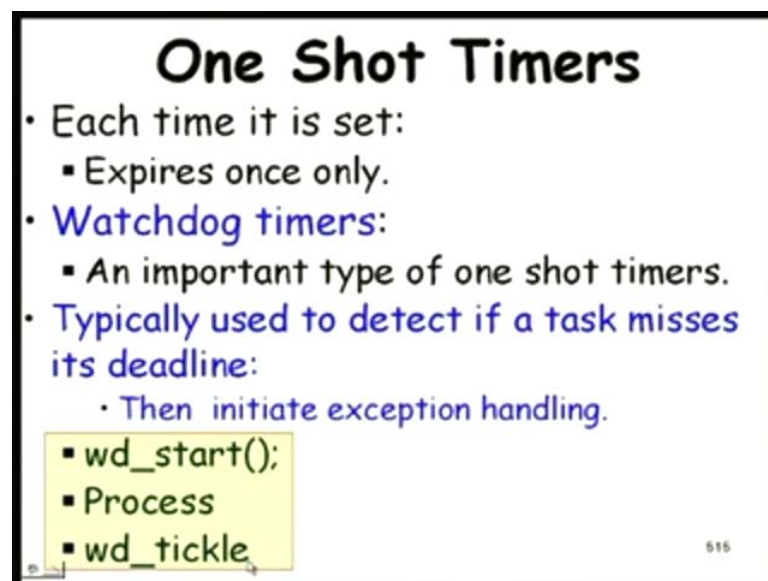
A periodic timer is typically used for sampling events at regular intervals like you want to do something every ten millisecond or hundred millisecond, they set the clock the periodic timer and then each hundred or ten millisecond it will keep on giving an interrupt, it based on that you would have the handler routine to handle the reading some sensor value and so on. So, that is one main use of a periodic timer.

Now, let us look at the periodic timer also, we will see what its main use and the timer is set once, the periodic timer is set once and once it is set, periodically it will keep on giving timer interrupts, and each time the periodic timer expires the handler routine will be invoke, and again it will be reinserted in the timer queue. So, the timer has expired

again in the timer queue, we are discussing the timer will be again reinserted it is expiration time updated.

So, these are the expired, and then the handler routine will be invoked, queued actually not invoked it will be queued and then again this same timer will be inserted back into the queue with updated expiration time.

(Refer Slide Time: 10:30)



One Shot Timers

- Each time it is set:
 - Expires once only.
- Watchdog timers:
 - An important type of one shot timers.
- Typically used to detect if a task misses its deadline:
 - Then initiate exception handling.

- `wd_start();`
- Process
- `wd_tick`

515

A one shot timer once it is set will expire only once, as the name itself says and these are popularly known as watchdog timers. So, very popular name of all real time programmers is a watchdog timer. Now, the watchdog timer is used very frequently in any real time programming, the main use is in determining whether a task missing its deadline and if the tasks misses its deadline then the handler routine with this timer will initiate some exception handling routine.

So, this is the typical structure of a real time task. So, first set the watchdog timer this is the system call, set the watchdog timer, let say that watchdog timer start for ten millisecond, so the watchdog timer will expire only once after ten millisecond, and if the process does not complete by ten millisecond then the handler routine here will be invoked, you have to attach a handler routine with a one set timer that we have not shown here just saying that watchdog start watchdog timer.

And if the process completes before the watchdog timer expires; that means, it has completed within its deadline, then we do not need the handler routine to be invoked just say `wd_tick`. So, this is the system call that is invoked to reset the watchdog timer. So, this is typically the structure of every task to check whether the task is completing within its required time otherwise an handler routine will be invoked.

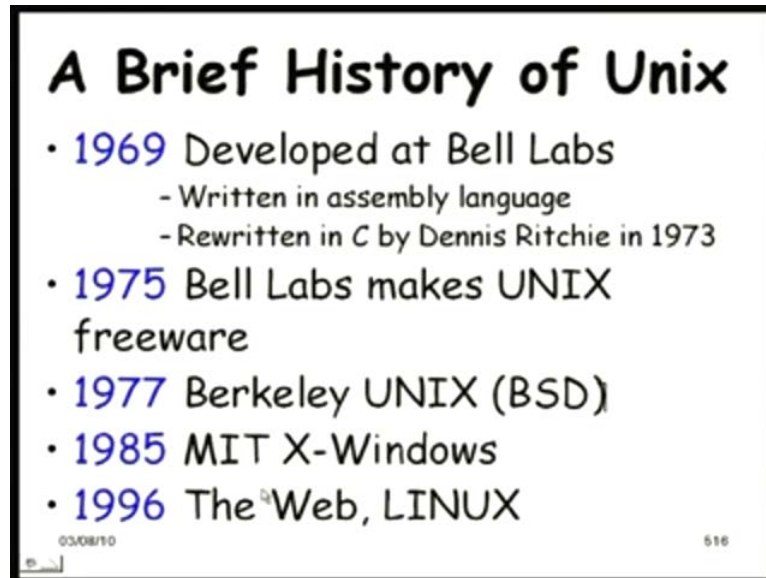
Here the timer is reset for some other purpose.

No, this is the `wd_start` is the watchdog timer which is set.

Now, if the task completes we do not need the handler routine to be invoked the exception handling need not occur. So, it has completed let us say 10 millisecond it is set and by 7 millisecond we find that the processing is over. So, then this system call will be given `wd_tick` which will take out the watchdog timer from the queue, it has completed within time, no exception handling is required the watchdog timer will be disarmed, but if the processing takes more than 10 second the watchdog timer will fire in the timer queue and the handler routine will be invoked.

Now, before we start discussing about the real time operating systems, we need to understand some basic issues on Unix, and these are slightly different from what you have done in your first level operating system course something other than that will be discussing, because we will see that many of these real time operating systems they are actually based on Unix variants basically.

(Refer Slide Time: 14:00)



So, if we look at the history of Unix developed at a t and t bell labs in 1969 written in assembly language and it was rewritten by Ritchie in c language in 1973.

But, bell labs is a telecom company they did not force you that they will use Unix commercially, and since it was a very good operating system and they did not have much use for it at that time they make it a freeware in 1975, there were many recipients or many could get the source code of Unix and then based on that many variants of Unix kept on appearing, the first one was possibly the BSD the Berkeley Software Distribution.

So, here the Unix are source code, the university of California at Berkeley they got a large project where they incorporated t c p i p into Unix, and they came up with the socket the BSD was the first support socket in Unix that was a of course a major improvement to Unix networking support on Unix, and then 1985 the x-windows was incorporated. So, all Unix distribution that you get you now have x windows in it is for the graphical interface, and then the web support and then the Linux appeared in 1996.


But, there were several over variations of Unix based on the freeware Unix in 1975 even a t and t kept and developing Unix, u 0 version 5 of Unix that became popular and then the other vendors like s c o s c o, Unix h p u x, all tricks from digital corporation each

then you have e y x from IBM and. So, on many variants of Unix appeared from 1975 version freeware that was made available.

(Refer Slide Time: 16:41)

Linux

- Unix was designed to be a mainframe operating system.
 - Could not run on PCs in 80s.
 - Of course, prior to 80386 virtual memory could not have been supported.
- Linus Torvalds at the university of Helsinki in 1991 :
 - Tried to use concepts of MINIX (Tanenbaum) to write a Unix operating system for PCs.



OS/161 617

And as the PC appeared 1980s, the PC could not run Unix it had a very simple operating system, why cannot Unix run on PCs 1980s. What do we think?

There were many reason why Unix could not run on the PCs, one reason is that you need some support for virtual memory in the from the processor it was not there in the processor before 80386 before 386, the hardware support for virtual memory was not there, what is the hardware support you need for virtual memory, can anybody answer this question, basic I mean first level question.

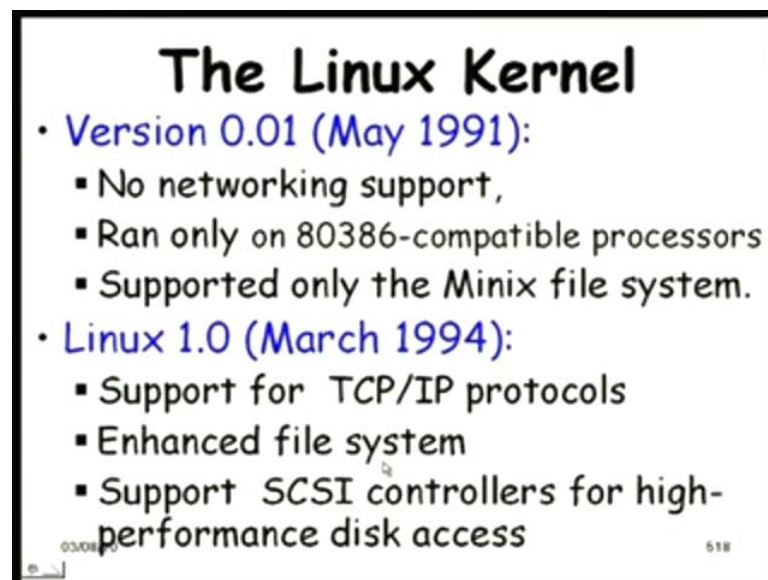
(())

You need the memory management unit and what are the functions of the memory management unit if it is asked that is a separate question, as he was saying t l b etcetera. So, the paging support has to be there and if you are not very sure about this, this is a very basic question, please look up what is the hardware support you need for virtual memory typically goes by the name memory management unit, but what are the different units in that memory management unit please check that up. And, also the p c was had the two little, I mean it runs on initially on floppy and then your hard disk could not really run Unix which was a large code. Unix was initially meant for servers or

mainframe computers actually, and since Unix did not run on PCs and by the end of 1980s that is 88 etcetera.

The PC had become powerful and then there were who were thinking of having Unix run on the PC. Linus Torvalds who was a student at university of Helsinki, he from the operating system book of Tanenbaum where the Minix operating system source code is there based on Unix actually mini Unix. So, he just may be use of that and develop the Unix for PCs and where the meaning in it was already supported in 80386 onwards 8486, and he could efficiently use it and then develop the basic skeleton on which a virtual memory operating system can be run on PCs.

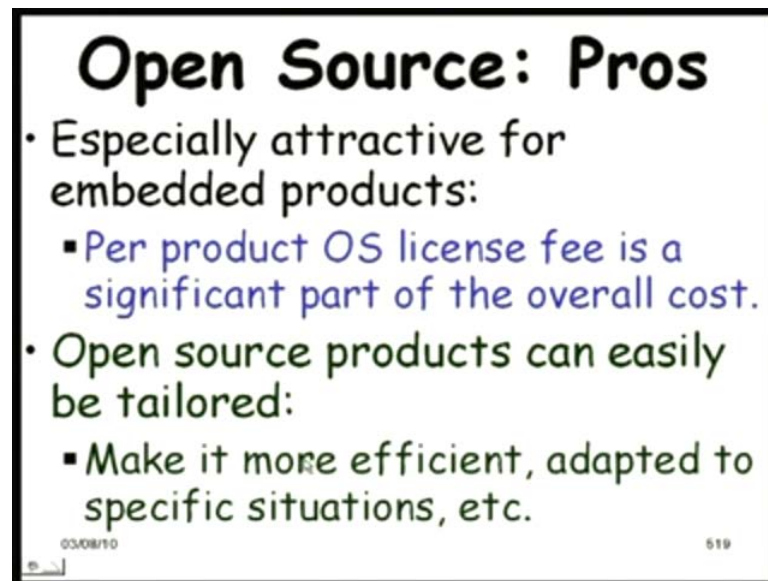
(Refer Slide Time: 19:50)



And from 1991 he made it available as a freeware, but remembers that the code is not really based on the freeware that was made public by a t and t corporation.

He had entire new code written based on the Minix operating system, but the initial version that he made it public, there is no networking support only in 80386 compatible processors and supported the Minix file system and from that point onwards, it had a very special type of development, where the development occurred by many collaborating programmers on the net. Slowly TCP IP protocol was added by somebody, enhanced file systems were added SCSI controllers were high disk high performance disk access was added.

(Refer Slide Time: 20:54)



And it becomes a freeware and it has so far played a very important role in the operating system area, the Linux operating system started from a student project or a student zone initiative, but before we look at the open source, we will see that open source real time operating systems are very attractive, many vendors are actually developing based on open source software open source operating systems, but before we give our hundred percent commitment to an open source.

Let's see the pros and cons, because definitely these are attractive and it would be nice to use this, but this also come with a bit of a penalty, now let us look at that, because we look at both the open source real time operating systems and also the commercial real time operating system.

One thing is that you say one license fee and when you talk of an embedded product costing few hundred rupees or few thousand rupees license fee is important. Now, even if you save ten rupees or fifty rupees on license fee your product becomes competitive a highly competitive embedded market.

The other advantage is that you have the code available to you can easily tailor it take some part, delete some part, modify parts can make the operating system more efficient for a specific application adopt a specific situation.

(Refer Slide Time: 22:41)

Open Source Success Stories

- **Apache:**
 - Runs more than 50% of world's web servers.
- **Perl:**
 - Most of the live contents on the web.
- **BIND:**
 - Provides DNS (Domain Name Service) for the entire Internet.

03/08/10 620

So, these are the advantages of an open source, and there are success stories based on this open source, for example the apache web server is almost the defective, anywhere you go and ask, what is the web server you are using, we run apache web server. So, the free server more than 50 percent of the world's web server runs Apache web server.

Similarly, we have Perl for live contents on the web the BIND which provides the DNS for the entire internet is a freeware.

(Refer Slide Time: 23:26)

Open Source OS: Cons

- Free OS can cost more for product development:
 - More time to develop device drivers
 - Can increase labour more than offset the commercial OS license fees saved
- Open source license model:
 - Requires to publish code developed on OS.
 - Some studies even show a recent decline in Open Source OS use.

03/08/10 621

But, there are also penalties one is that it costs more for development, why should it cost more for development, because you might have to you know new device you are trying to use the device driver would not be available until somebody make makes it open source and you possibly have to develop the device drivers, and then the development might be more time consuming here in the open source, you need to understand finding open source programmers may be more difficult, we need to understand the internals of this.

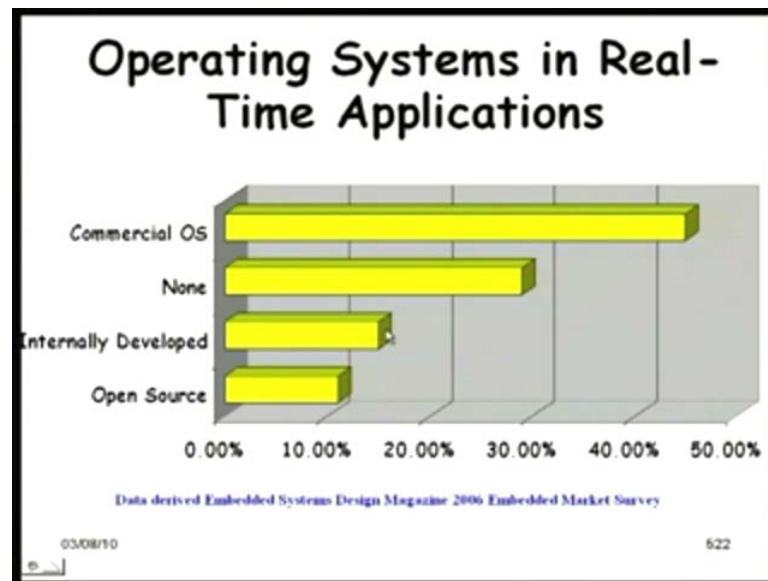
So, the labor cost may be more and difficult to find skilled man power open source, there is another factor that we need to remember is about the open source license model, it is not that you just take the open source and keep on using it, but you have to avoid by a license model which requires that if you make any modifications to the code need to publish that.

So, any development, for example free operating system, if you change it, tailor it and so on you need to again make it freeware that is the understanding, and if we are a commercial manufacturer and somebody says that you make your software freeware definitely it will be a big turn off, so then what is the advantage of using this compotators will get my software.

So, there is a pessimistic result here I mean I do not know that authenticity of the result, but says that there is a decline in operating system open source operating system use, but again there are some other report saying that there is a big increase in the real time Linux as a real time operating system, and we know that Linux has become very popular, because operating system is a significant part of the desktop cost.

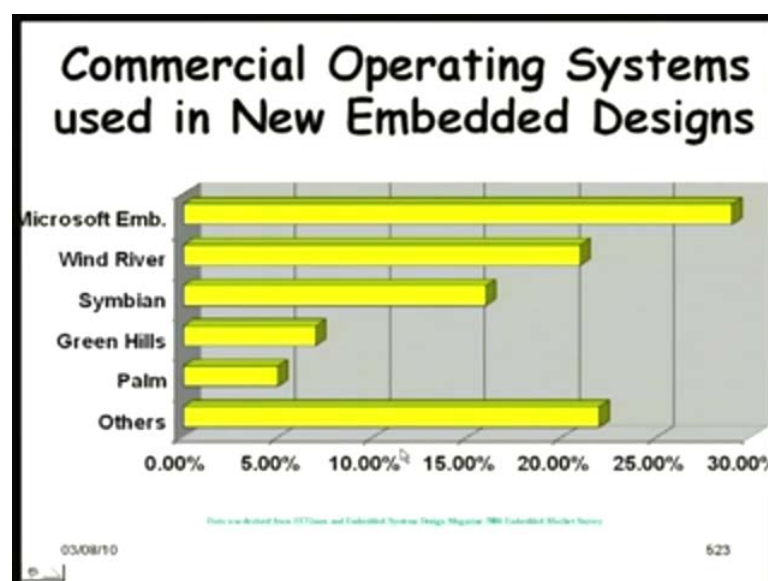
So, as for as desktop is concerned we can see now it now a day's many desktop that run only on Linux the cheap ones. So, that is for the debatable and these are controversial reports let us not get into that, but at least let us be aware of the issues before we proceed further.

(Refer Slide Time: 26:08)



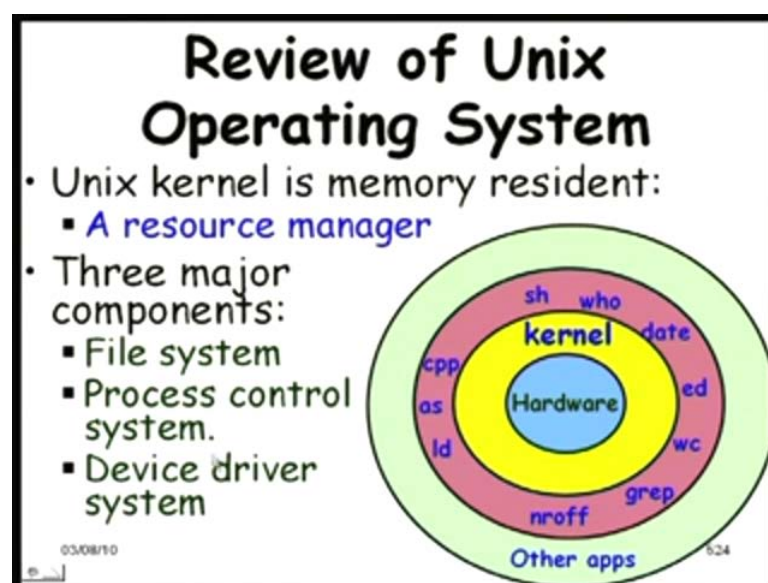
So, let us look at the real time situation operating system situation here. So, this is again a statics published in embedded system design magazine says that the commercial operating system that the priced one constitute nearly 50 percent slightly more than 40 percent of commercial operating system, and then there are many of these applications which are none of the operating system internally developed operating system is about 15 and then open source is only about 10 percent.

(Refer Slide Time: 26:48)



And among the commercial operating systems if we look at the embedded systems, we will see that the Microsoft's c operating system Microsoft embedded or the Wind c that has the largest following more than 25 percent here, and then the wind river is a big player as we proceed with our discussion, we will discuss about the wind river operating system, the Symbian operating system, green hills, the palm OS etcetera and then there are also many other players as we are saying which have use as in some specific situations.

(Refer Slide Time: 27:36)

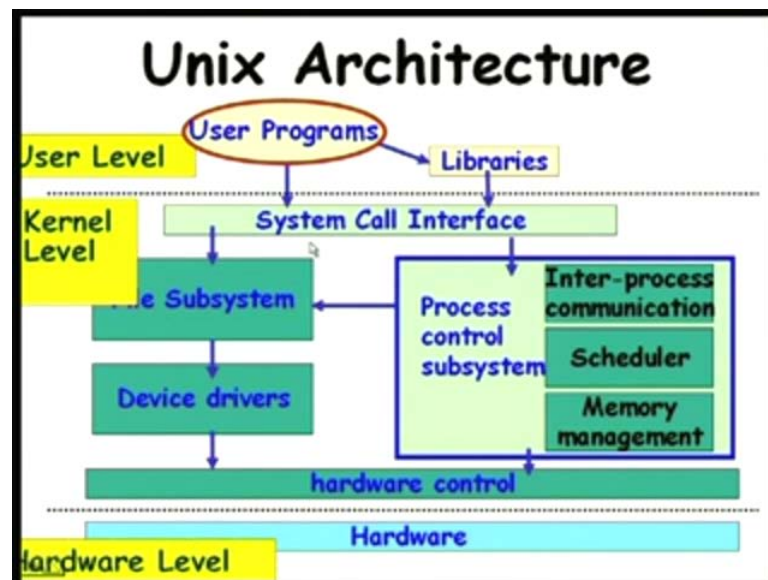


Now, as we are saying that we need to understand some very basic issues on Unix operating system and then we need to understand why Unix as it is not a suitable real time operating what is lacking there internally, and then based on that knowledge we can appreciate the real-time operating system. So, this is a very basic concept we know that any kernel when we talk of what is an operating system kernel. A kernel is the memory resident portion of the operating system.

The kernel is memory resident and it is basically the manager of the resources, the hardware resources, the data resources and so on. And, the operating system if we look it there are the kernel; there are three main components of this, the file system, the process control system and the device driver system and the memory management of course,

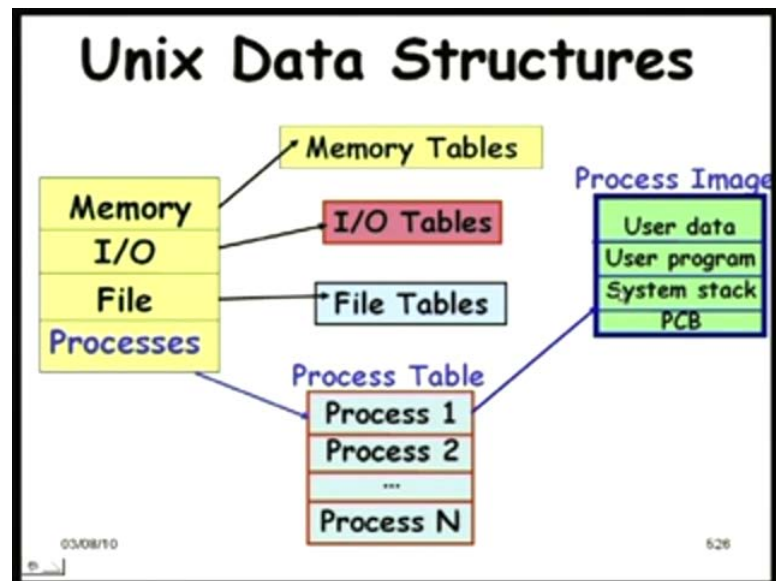
So, I think we it would have been better if we wrote file system, process control system and the memory management system. So, this will be the file system and the memory management system, and then if you look at the process control system we have the scheduler here.

(Refer Slide Time: 28:54)



We have put the memory management under the processor control system inter-process communication. So, the user program, they have some libraries or the system calls, these are the library supported by the operating system and then as we are saying that once we invoke a operating system library, it traps into the kernel mode through a software interrupt. The mode changes from a user mode to a kernel mode through a system call interface.

(Refer Slide Time: 29:59)



And, then there are various data structures that are maintained, for example there will be various memory tables that need to be maintained, which memory, which page is been used, how frequently all those things.

I O tables I Os that are been used, who is using etc, the file tables, which files are open, what is the current access block of the file etc, the processors for every process, process tables are maintained, for example we have the process images that are maintained the user data user program, the system stack that is used on the control block that is being used for this process control block.

(Refer Slide Time: 30:49)

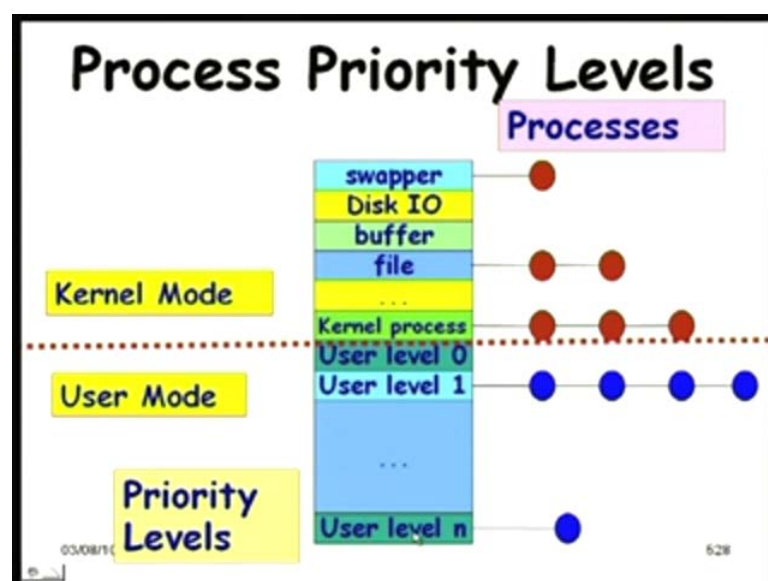
System Call

- How does a process access system resources?
 - Through system calls.
- Example system calls:
 - **Process control**: fork, wait, exec, exit
 - **File system**: open, read, write, lseek, close, chdir, chown, chmod, stat, fstat
 - **Others**: dup, mount, unmount, link

03/08/15 627

Now, how does a system access its resources through a system call this we were discussing. A user process can access system resources through a system calls, a user process cannot directly manipulate the resources. It can only make calls to the operating system and there are many examples of system calls, for example fork, wait, exit, exec etc. File system calls open, read, write, lseek etc. There are other calls like a pipe call dup, then mount device etc. Unmount, link.

(Refer Slide Time: 31:40)



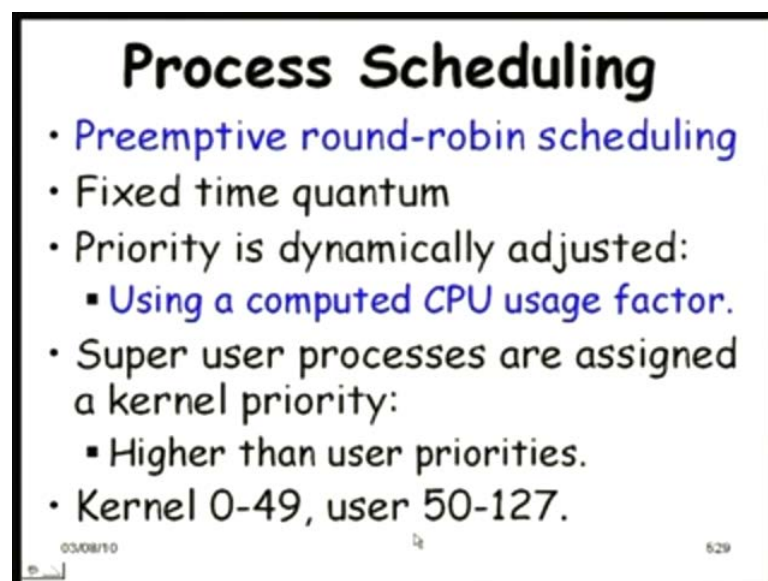
And we will discuss this diagram with a much more elaboration, but these are basic issue here, just see here if we look at the priorities levels in a Unix operating system see that there are two modes.

The user mode and the kernel mode, and the user level there are various types of priorities; user level 1, user level 0, to user level n, n is the lowest priority 0 is the highest priority, and there are tasks processes queued at various levels, there will be context among these levels of tasks.

And then in the kernel mode there are also various priority levels, all kernel priorities are higher than the user priorities, just check here these are two different bands and then among the kernel also there are various priorities. The kernel processes are possibly the lowest priority and then we have file IO, buffer IO, buffer management, disk IO, swapper has the highest priority.

We'll later on understand why really these kind of priority arrangement is necessary among the operating system processes and then there are various tasks that are queued at various priority levels, there is a basic picture we will remember it and then we will elaborate on this as we proceed with our discussion.

(Refer Slide Time: 33:22)



Process Scheduling

- Preemptive round-robin scheduling
- Fixed time quantum
- Priority is dynamically adjusted:
 - Using a computed CPU usage factor.
- Super user processes are assigned a kernel priority:
 - Higher than user priorities.
- Kernel 0-49, user 50-127.

03/08/10 629

The scheduling policy that is followed is a round-robin scheduling with a fixed time quantum, and the priority is dynamically adjusted that is what we are saying, the Unix by

itself does not support any static priority as the task keeps on running it is priority value keeps going up and down based on what the task is doing.

So, what the scheduler does is that it computes a CPU usage factor that is how much fraction of its time or slice it could use the CPU. If it run through the entire slice then its priority will decrease, if it could use hardly much of the CPU slice, because possibly it had to block for I O or something its priority will increase. If it uses the full slice time slice allocated to it its priority will decrease. If it could not use its time slice, because of possibly it had to wait for I O or some event its priority will increase.

We'll see exactly how it computes this, because this is importance for us we will see how the Unix computes this the dynamic priorities, we will see that it uses a waited history it is not that what it did in the last slot last slice what it did, but what it did in the last, last to last and before that the history is used for computing the usage factor, and then the super user processes are run at kernel priority which are more than the user priority, the kernel is typically 0 to 49 and the user is 50 to 127.

(Refer Slide Time: 35:21)

Entry into the Kernel

- **Synchronous:** Kernel performs work on behalf of the process:
 - **System calls:** Through UNIX API
 - **Hardware exceptions:** Arise due to some unusual action of the process.
- **Asynchronous:** Kernel performs possibly tasks unrelated to current process.
 - **Hardware interrupts:** Example: I/O completion, status change, real-time clock etc).

OS/08/10630

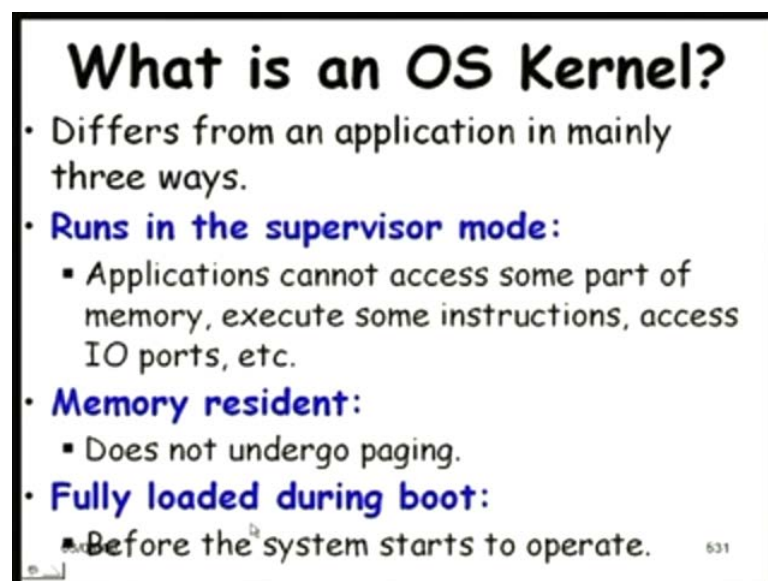
The entries to the kernel can either be synchronous or asynchronous. In a synchronous entry the kernel performs work on behalf of the process, for example make a system call that is a synchronous one or some hardware exceptions, let us say divide by 0 error

exception need to be handled. So, these are the synchronous ones, and these occur in the context of a specific process.

See, this hardware exception occurred on behalf of the process, wherein the process was running and the exception occurred and similarly the system calls made by the process. So, in the synchronous one where the current task waits for the operating system to do something is asynchronous entry into the kernel we can also have asynchronous entry to the kernel.

Where the kernel performs task unrelated to the current process for example, hardware interrupt where the kernel takes up the responsibility of processing is asynchronous entry into the kernel where it can run programs that are unrelated to the current process, for example I O completion interrupt, status change interrupt, clock interrupt etc.

(Refer Slide Time: 36:50)



What is an OS Kernel?

- Differs from an application in mainly three ways.
- **Runs in the supervisor mode:**
 - Applications cannot access some part of memory, execute some instructions, access IO ports, etc.
- **Memory resident:**
 - Does not undergo paging.
- **Fully loaded during boot:**
 - Before the system starts to operate.

631

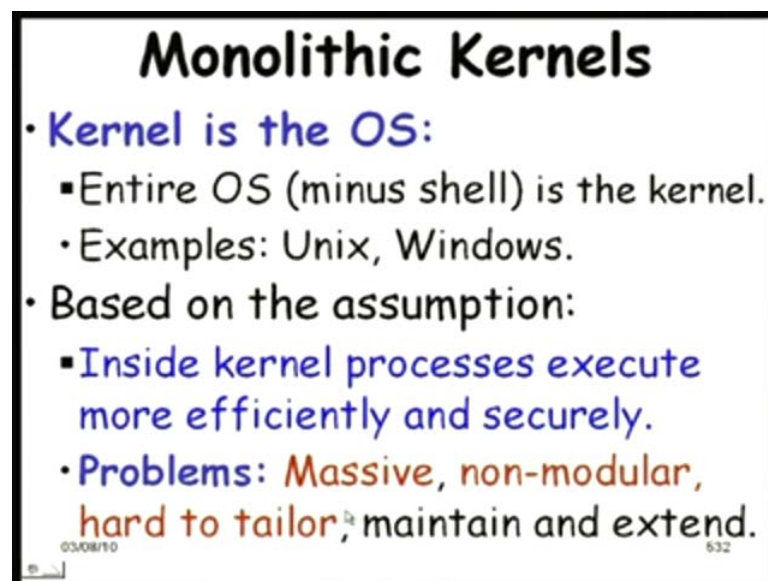
And we said that the OS kernel mainly differs in three principle ways from a traditional application, a user programs runs in three main ways.

The first one is that; it runs in the supervisor mode, the kernel programs, kernel processes they run in the supervisor mode that is they run at higher priority and also they can do privileged operations which the user programs can do, and the paging is not applicable to the kernel these are memory resident. The user programs undergo paging and these get fully loaded during boot and a resident, and also they can do some privileged operations

run at higher priority, and run do some privileged operation, like they can access the memory entire memory execute some instructions access I O ports etc.

They do not go undergo paging and they are fully loaded even before the system becomes ready to operate. It is a three principle differences we will keep in mind between a user level process and a kernel process.

(Refer Slide Time: 38:18)



Now, as we will discuss the different types of kernels, we will see that there are 2 main variations; one is the traditional monolithic kernel, in the monolithic kernel the principle or the concept here is the kernel is the OS, so talk of the kernel OS its almost the kernel minus of course, the shell etc.

It is the entire OS is the kernel. The idea behind a monolithic kernel is that the kernel processes execute efficiently and securely once there in the kernel mode. So, it would be advantageous to keep as many processes and task in the kernel mode as long as they are required to do some important tasks, it is advantageous to have them in the kernel mode that is the principle that is used here. But, the main problem with a monolithic kernel why this is going out of scope or its getting outdated, we will see the all later operating system kernels are based on micro kernels.

OS has many other functionalities like projects management, file management all those things. What is that kernel part of OS? What it is used for?

So, far we have already said that, see the kernel is the resource manager, it handles the processes including scheduling, communication etc. it handles memory which is important resources, it handles the devices, files these are important responsibilities of any kernel.

What is OS 2?

And then we have other things for example, a shell the user interface, we might have a compiler would you call it part of the OS, but it can also come with the OS. So, the kernel does not include the compiler you said that, but it also comes with the OS. So, that is actually in the shell and shell utilities.

We can say shell is an application.

Shell is an applications it is not really kernel you are right. So, the kernel is the resource manager which is resident, compiler need not be resident in the memory.

Sir OS is kernel plus application.

Of course, it has the kernel plus some shell and shell utilities.

The shell the outer part of the OS is not really the kernel.

Windows is monolithic.

Windows Unix etcetera all these are monolithic, we will discuss about this in more detail.

The entire part of the operating system is resident in (()).

We will discuss about this. So, the disadvantage we are saying that see monolithic is becoming outdated, that concept developed long back and the principle was that as long as these become part of the OS they can do some privileged operations.

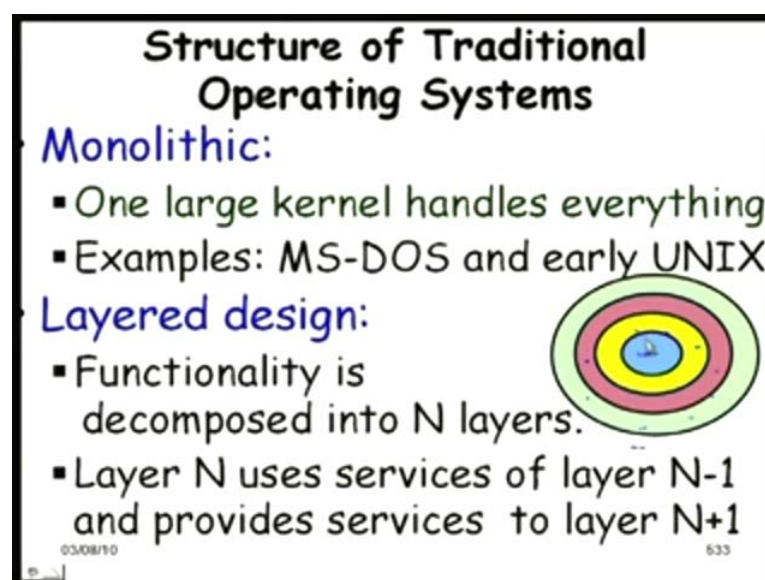
They execute more efficiently and securely, because the user mode processors cannot really affect them, the user mode processors run at a lesser priority and also they are not privileged like this, but that thing is becoming outdated we have most of the modern

operating systems including the real time operating system, micro kernel operating systems, the kernel is kept to the minimum that is why it is microkernel.

Now, let us find out what is the problem with this monolithic kernel, one is that it is massive does many things all the resource management activity, non modular see all of them they have access to each other data, because all of them are privileged they can access anything they want and hard to tailor, because once it is in the kernel mode, you cannot really hard new things or take out some of these things without really recompiling your kernel. Kernel recompilation will be necessary even if you want to make a small change to the kernel even if you want to add a new device driver need to recompile your kernel.

These are difficult to become difficult to maintain and extend of course, this goes with this, because hard to change it and extend yes you cannot really hard without recompiling the kernel, and also since, these execute in the kernel mode debugging the kernel is very difficult those who have worked with kernel they might know that, because you know how do you find out who has changed to which part of the code, because all of these, these are non-modular code remember.

(Refer Slide Time: 43:57)



The traditional operating systems are monolithic one large kernel handles everything, we are saying that Unix and DOS. See, now we will see that slowly the windows and Unix

they are becoming microkernel based, and the typical construct of this traditional operating system, so even read in the first level course that they are a layered one, functionality is decomposed into N layers and each layer provides service to the layer above it and utilizes the service of the layer below it. That is the typical structure of an operating system we studied in the first level course,

(Refer Slide Time: 44:52)

Microkernel Approach

- **Minimalist kernel approach:**
 - Run as much as possible in application space.
 - Kernel is modular and small.
 - 10KBytes to a few hundred Kbytes.
 - Easier to port, maintain and extend.
 - No agreement on what should be in kernel.
 - Typically process management, memory management, IPC.

03/08/10 534

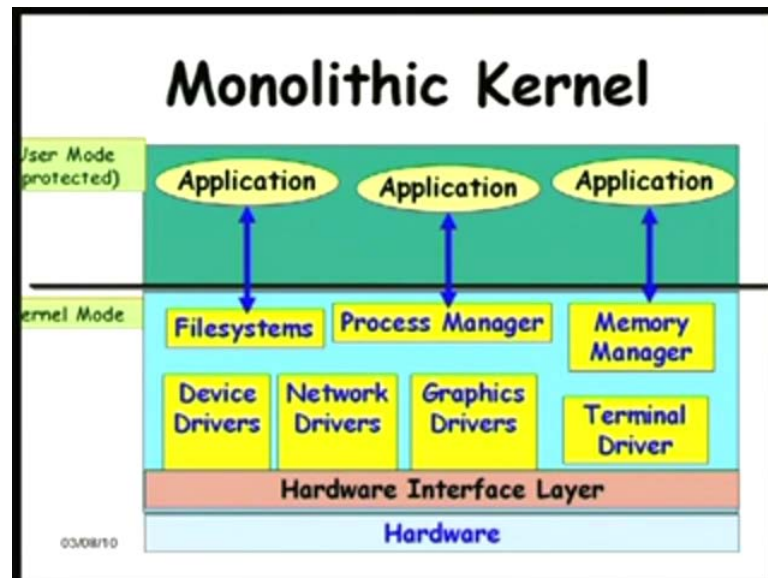
But, we will see that the microkernel approach is become popular now, it is the minimalist kernel approach as the name says it is a microkernel. The idea is that run as much part of the kernel traditional kernel in the application space as a user program.

The kernel is modular and small from 10 kilobytes to a few hundreds of kilobytes is the kernel, it is easier to port maintain and extend, because all others are user programs you can just keep them, adding them and taking them out the other part of the operating system without really having to compile the kernel, stop your operating system and compile the kernel is not necessary, but we will see that in the microkernel approach you will find huge literature on this, but you will see that there is no really general agreement on what should be in the kernel.

But, typically we will see that the process management, memory management and some inter process communication is part of the kernel, but we have also situations in

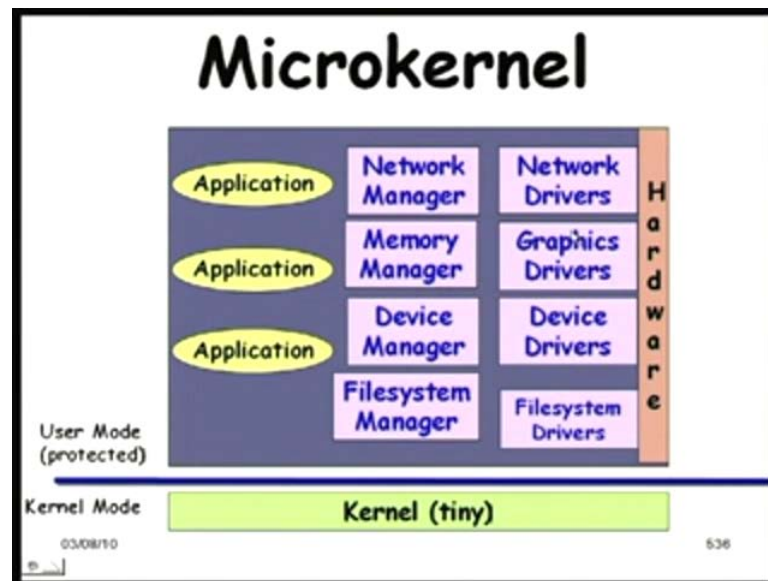
microkernel where only the basic process handling facilities are there, rest everything is in the user space we will see that.

(Refer Slide Time: 46:25)



So, if you look at the monolithic kernel approach, you see that the operating system all of these are in the kernel mode starting from device driver to network driver to graphics driver, terminal driver, memory manager, process manager, file manager everything is in the kernel mode which directly interact with the hardware interface layer, we called it as the BSP or something last time board support package and this interact with the hardware and then the applications run on the user mode or user space, but they can make system calls to access the operating system service.

(Refer Slide Time: 47:15)



In the micro kernel approach we have a tiny kernel here, possibly it have some very basic process management, but see here process management and we are saying that the typically accepted functionalities or basic process management and some memory management that is what we are saying, but see here all these are running in the user space, the network drivers, graphic drivers, device drivers, file systems, the file system manager, memory manager, network manager, device manager all these are running on the user space and then the applications are invoking them services directly without a system call. We will elaborate on this model later on, so this is another view of that.

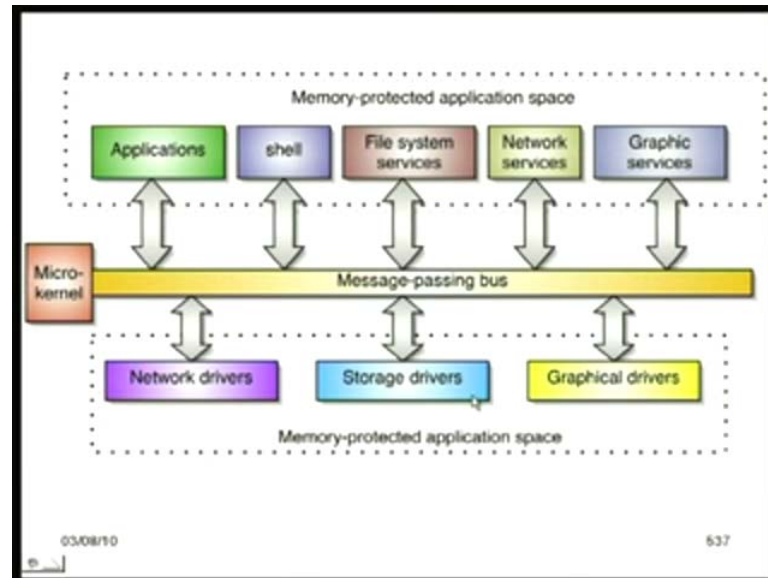
What is there in the Kernel part?

As we are saying that some very basic process handling, process creation, process scheduling etc and also very basic memory functionality are here. So, that is the accepted view of what is a microkernel, the rudimentary process management and some part of the memory management.

(()):

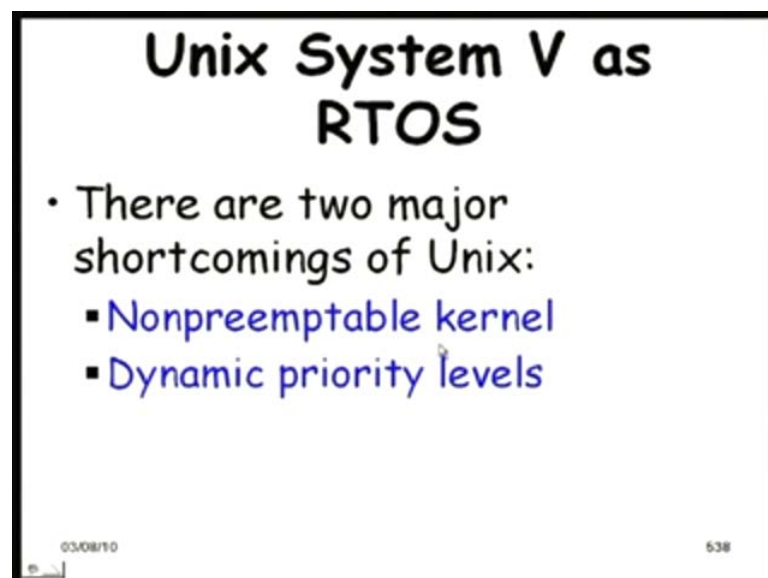
Yes exactly.

(Refer Slide Time: 48:56)



This is another view of the microkernel where we have the microkernel and there are various types of memory protected applications here, we know that all these are in the user space also they are memory protected and they just keep on communicating with each other through a message passing bus, this is a high level view of the same thing.

(Refer Slide Time: 49:20)

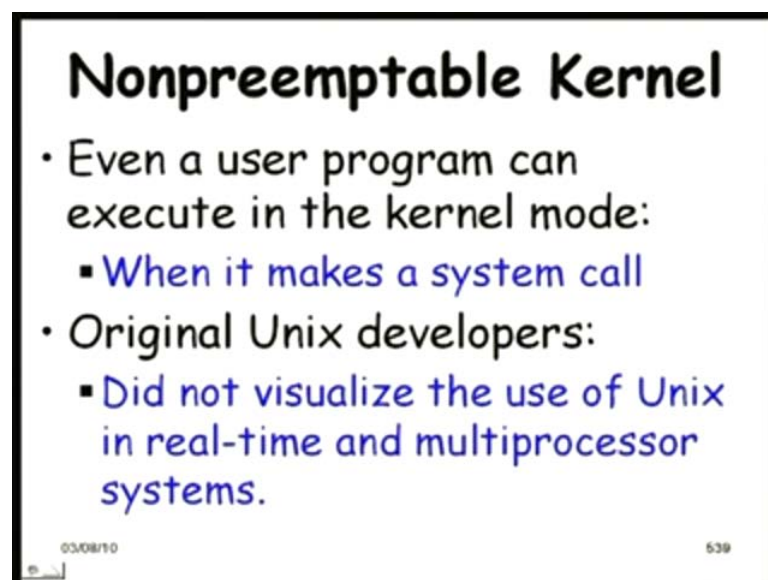


Now, with that basic idea, basic concept on operating systems microkernel and monolithic kernel and the basic services and so on. Let us, see why Unix is not suitable for a real-time operating system from the internal point of view not like that it you

cannot really run real-time schedulers, static priority level not available etc or some other facilities not available, we will see internally what is holding Unix from being used as a real time operating system.

So, if we analyze the reasons we will see that there are 2 main stumbling blocks, other things to quite satisfactorily or to some extent they can be taken care of other facilities I mean other feature of Unix can be extended, but there are 2 main stumbling blocks one is a nonpreemptable kernel and other is dynamic priority levels. So, let us check why the Unix is nonpreemptable in the first place, and what is the difficulty to make it a preemptable, and then why this dynamic priority levels cannot we just have static priority levels in the Unix and what does it do with this dynamic priority levels.

(Refer Slide Time: 51:13)



Nonpreemptable Kernel

- Even a user program can execute in the kernel mode:
 - When it makes a system call
- Original Unix developers:
 - Did not visualize the use of Unix in real-time and multiprocessor systems.

03/08/10 539

Why does it need in the first place, how does it help. So, let us understand those issue, because if you understand this we can very well understand the real time operating systems, and how they have developed and what is there inside that, how they are different from the Unix. Let us look at the non preemptable kernel part, we had said that even a user program can execute in a kernel mode by making system calls, and the original Unix developers did not visualize the use of Unix in real time and multiprocessor system.

So, what they said is that, when it is running in the kernel then it is the operating system or the one that is most privileged one is running it need not be interrupted, the user programs can be interrupted, but the kernel let it complete and then it can examine the interrupts etc.

So, the simplest thing they could do was to disable the interrupts once in the kernel mode, the idea was that if they allow the interrupts even when it is in the kernel mode when they will have to do extensive locking of the kernel data structures, otherwise the kernel data structures will lose their integrity, like let us say a one interrupt has occurred another interrupt is handled or let us say one process has not yet been created and even to create another process and so on.

The data structure will become inconsistent and locking would make it inefficient. So, why go for this inefficiency it is not required, because after all the kernel is the privileged one it, so not be interrupted.

(Refer Slide Time: 52:52)

Nonpreemptable Kernel

- In a real-time application
 - A nonpreemptable kernel can cause deadline misses.
 - task preemption time = time spent in kernel mode + context switch time
 - This can be of the order of a second in the worst case.

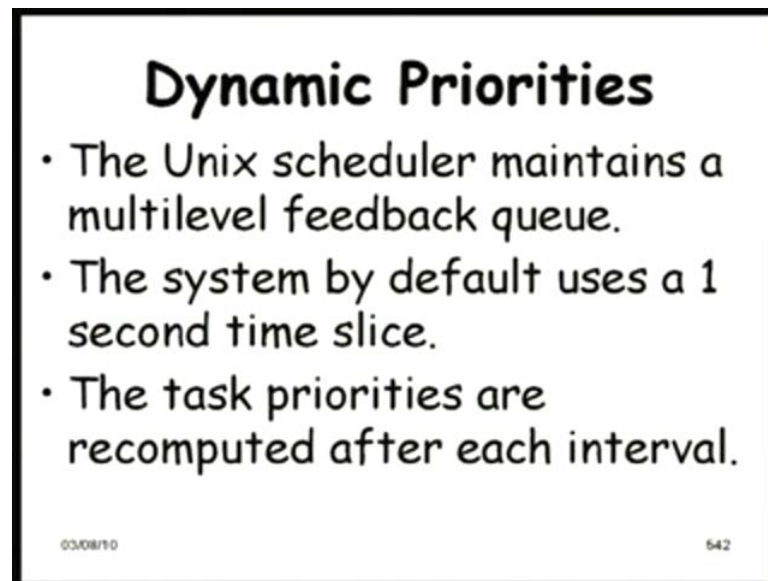
©3/08/10 541

But when we talk of a real time application, it would lead deadline misses, because the task preemption time will become the time spent in the kernel mode that is once you made the system call, it will execute in the kernel mode and then the context switch time once it completes the kernel then only the context switch can occur, and the time spent in

the kernel mode can be of the order of a second, for example it has initiated a read call or a write call can take a second or may be several seconds.

So, that much time the task cannot be preempted and by that time a real-time task miss its deadline.

(Refer Slide Time: 53:40)



Dynamic Priorities

- The Unix scheduler maintains a multilevel feedback queue.
- The system by default uses a 1 second time slice.
- The task priorities are recomputed after each interval.

03/08/10 542

Now let us look at the dynamic priorities I think we are coming to adjusting our time today. So, since this is an important issue the dynamic priority handling in Unix which we need at least 15 minute discussion.

It is a important topic here why dynamic priorities are needed, what is the advantage and do we need this in the real time situation, we will see that we will also need in real-time some real-time operating system dynamic priority levels, and we will understand what is the advantage of this and disadvantage of this and how it is supported, how exactly the dynamic priority values are computed, because the same formula is used occurs all operating system traditional operating systems. So, come to a logical point we will stop here we will start our discussion from this point onwards in the next class thank you.